Programação de Sistemas

Acesso a ficheiros – panorama global

Introdução (1)

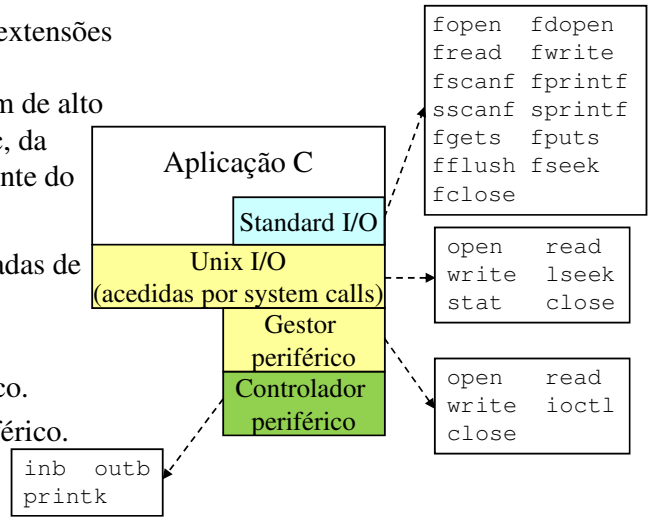
[Definição]: No Linux, um **ficheiro** é uma sequência de Bytes.
Os ficheiros são catalogados no Linux em 3 tipos:

- Regulares (-)
- Directórios (d), com estrutura interna de dados em árvore
- Especiais, sequências residentes fora de unidades de massa
  - Dispositivos de caracteres (c) ou de bloco (b)
  - Ligações (l)
  - Tubos (p)
  - Socket (s)

- A todos os ficheiros são disponibilizados o mesmo conjunto de funções (open, read,...) – embora algumas possam gerar erro: ex, ler da impressora.

Introdução (2)

- Os ficheiros são acedidos, de forma distinta, em diversos níveis de abstracção.
  - Aplicação: chamadas de extensões definidas na biblioteca normalizada de linguagem de alto nível (ex: biblioteca glibc, da linguagem C), independente do sistema operativo.
  - Nível de processo: chamadas de sistema operativo.
  - Para dispositivos:
    - Gestores de periférico.
    - Controlador de periférico.



Introdução (3)

- A biblioteca C é usada em sistemas operativos distintos (Lunix, MSDOS,...)
- O Unix I/O pode ser acedido por qualquer linguagem de programação oferecida pelo Unic (C, f77,...)
- O Linux oferece uma terceira forma de acesso a ficheiros, `mmap` – que mapeia dispositivos (incluindo ficheiros) à memória virtual.
  O `mmap` é abordado numa secção especial do capítulo de gestão de memória central.

Biblioteca normalizada do C (1)

- Biblioteca normalizada do C99 (versão normalizada do C definida pelo ANSI em 2000), dividida em 24 partes (15 pelo C89).
  - Funções e constantes definidas no Linux em /usr/include/stdio.h
  - Disponibilizada no Linux pelo arquivo libc.a
- Os ficheiros são manipulados com base em ponteiros para `FILE`.

```
typedef struct {
        char*   _ptr;
        int     _cnt;
        char*   _base;
        int     _flag;
        int     _file;
        int     _charbuf;
        int     _bufsiz;
        char*   _tmpfname;
} FILE;
```


Biblioteca normalizada do C (2)

A. Funções de acesso

FILE * fopen(char *, char *);

Abre um ficheiro, em que

- 1º parâmetro - localização do ficheiro (a partir do directório corrente)
- 2º parâmetro - modos de acesso.

Modo	Acesso	Posição inicial	Se existir fich?	Se não existir fich?
"r"	Apenas leitura	Início		Retorna NULL
"r+"	Leitura e escrita	Início		Retorna NULL
"w"	Apenas escrita	Início	Conteúdo rescrito	Ficheiro criado
"w+"	Leitura e escrita	Início	Conteúdo rescrito	Ficheiro criado
"a"	Acrescento	Fim		Ficheiro criado
"a+"	Acrescento e leitura	Fim		Ficheiro criado

Biblioteca normalizada do C (3)

- Linux os processos de utilizador são lançados com 3 ficheiros abertos
  - stdin: *standard input* (usualmente associado ao teclado)
  - stdout: *standard output* (usualmente associado ao terminal)
  - stderr: *standard error* (usualmente associado ao terminal)
- As operações de escrita/leitura do ficheiro dependem da memória tampão ("buffer") usado
  - "Line-buffered", ou "text-stream", formado por linhas até 256 caracteres-cada uma terminada obrigatoriamente por '\n'.
  - "Fully-buffered" e "Unbuffered".

```
int fclose(FILE *);
```

Fecha ficheiro, descarregando ("flush") dados que ainda estejam em memória tampão ("buffer") em que

- 1º parâmetro - referência do ficheiro a fechar.
- Retorno – 0 em caso de sucesso, EOF em caso de erro.

Biblioteca normalizada do C (4)

B. Funções de leitura

- Ficheiros de texto:

```
int fscanf(FILE *, char *,...);
```

- 1º parâmetro - referência do ficheiro a ler.
- 2º parâmetro - formatação da entrada
- Parâmetros seguintes - localização dos dados a ler.

Nota: scanf(char *format,...) é equivalente a fscanf(stdin, char *format,...)

Opções de formatação

Opção	Formato
%d	Inteiro com sinal
%f	Vírgula flutuante com sinal
%x	Hexadecimal sem sinal
%b	Binário sem sinal
%c	Caractere
%s	Cadeia de caracteres

Biblioteca normalizada do C (5)

int fgetc(FILE *);

Retorno – caractere lido, convertido para (int).

Nota1: tentativa de ler para além do fim do ficheiro retorna EOF

Nota2: getc() equivalente a fgetc()

Nota3: int getchar() equivalente a int getc(stdin)

char *fgetcs(char *, int, FILE *);

Lê sequência de caracteres até ao número indicado no 2º parâmetro, terminando antes se for detectado EOF.

Nota: `gets()` corresponde a `fgets()` de STDIN, mas lê até EOF ou \n sem se preocupar com a dimensão da memória tampão. Logo, evitar usar `gets`.

int ungetc(int, FILE *);

Reenvia caractere para o ficheiro.

Biblioteca normalizada do C (6)

- Ficheiros binários

size_t fread(void *, size_t, size_t, FILE *);

Lê de um ficheiro, em que

- 1º parâmetro - localização da zona de memória
- 2º parâmetro - dimensão dos elementos a ler.
- 3º parâmetro - número de elementos a ler.
- 4º parâmetro - referência do ficheiro a ler.
- Retorno – número de elementos lidos.

Biblioteca normalizada do C (7)

C. Funções de escrita

- Ficheiros de texto:

```
int fprintf(FILE *, char *,...);
```

Nota: printf(char *format,...) é equivalente a fprintf(stdout, char *format,...)

```
int fputc(int, FILE *);
int fputs(char *, FILE *);
```

- Ficheiros binários:

```
size_t fwrite(void *, size_t, size_t, FILE *);
```


Biblioteca normalizada do C (8)

D. Gestão da memória tampão

- Memória tampão, gerida automaticamente pelas funções da biblioteca normalizada
  - existe na área do utilizador e possui dimensão de 1 bloco de disco.
  - memória tampão é criada na função fopen()
- A escrita para (leitura do) dispositivo realmente feita apenas quando a memória tampão está cheia (vazia).
  Nos ficheiros de texto, escrita e leitura realmente feitas quando for detectado o '\n'.

- Definição de buffer de utilizador

```
void setbuf(FILE *, char *);
```

- Descarga de dados em buffer de utilizador:

```
int fflush(FILE *);
```

Biblioteca normalizada do C (9)

E. Reposicionamento
   - Em FILE é mantida a posição do ficheiro, de tipo long int
   - Na abertura, FILE é posicionado no início (fim) nos modos "r" e "w" (modo "a").

   • Recolha da posição corrente (no início do ficheiro, o valor é 0)

```
long int ftell(FILE *);
```

   • Reposicionamento no início do ficheiro

```
void rewind(FILE *);
```


Biblioteca normalizada do C (10)

   • Reposicionamento aleatório

```
int fseek(FILE *, long int, int);
```

Desloca posição corrente para localização de partida+nBytes (garantido apenas em ficheiros binários).

   - 3º parâmetro - localização de partida:
     SEEK_SET (0) = início de ficheiro
     SEEK_CUR (1) = posição corrente
     SEEK_END (2) = fim do ficheiro
   - Se 2º parâmetro for positivo (negativo), deslocamento é dirigido para o fim (início) do ficheiro.

Nota: garantido apenas em ficheiros regulares

Biblioteca normalizada do C (11)

F. Ficheiros temporários
   - O ficheiro temporário possui as seguintes características
     - Identificador único, desconhecido do utilizador.
     - Ficheiro de tipo binário acessio escrita, "wb+"
     - Automaticamente eliminado no fclose().
     - Usado tipicamente para armazém temporário de dados de grande dimensão

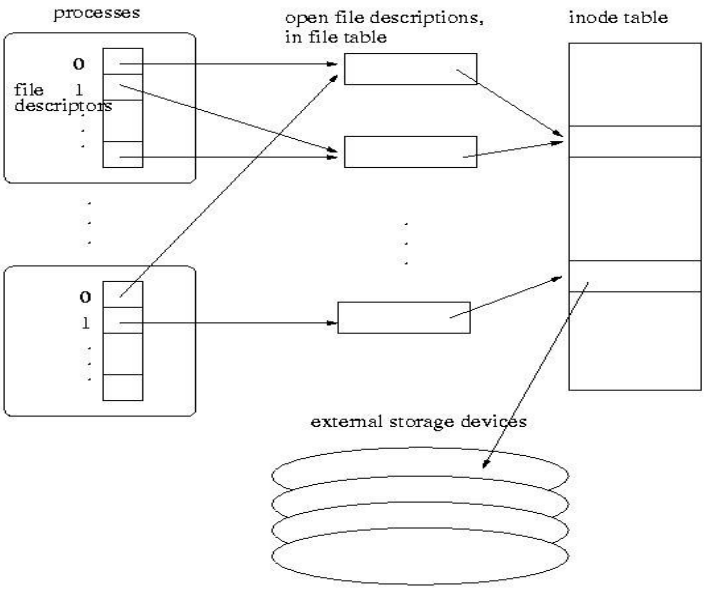
   - Criação de ficheiro tenporário

```
FILE *tmpfile(void);
```


POSIX (1)

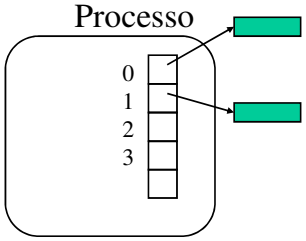
- Ficheiros são geridos no POSIX por 2 estruturas de dados
  - Descritor de ficheiro : inteiro positivo, único em cada processo, que referencia entrada na tabela de descritores.
    No Unix, os descritores de ficheiros podem referir
    - Directórios
    - Dispositivos de bloco (disco, disquete,...) ou de caracter (terminais), Sockets,
    - pipes ou FIFOS.
  - Descritor de ficheiro aberto : existente no espaço do núcleo
    - Liga os descritores de ficheiro (privados aos processos) ao bloco de controlo do sistema de ficheiros.
    - Mantém a posição corrente de leitura/escrita.
- No UFS, o bloco de controlo designado i-node contém
  - informação administrativa (dono, datas, permissões de acesso,...)
  - localização dos dados.

POSIX (2)



POSIX (3)

A. Operações na tabela de descritores de ficheiros
POSIX define diversos criadores de descritores de ficheiros, conforme o tipo de ficheiro.
   - Número máximo de descritor de ficheiros abertos indicados pelo comando Shell `ulimit -n`.
   - No lançamento de processo, o Shell cria descritores de ficheiros nos 3 primeiros índices:



Índice	Nome	Fich. por omissão
0	`stdin` (standard input)	Teclado
1	`stdout` (standard output)	Monitor
2	`stderr` (standard error)	Monitor

- A função `scanf` lê dados sempre do descritor `0`.
- A função `printf` escreve dados sempre no descritor `1`.

POSIX (4)

- Abertura de dispositivo de caracteres ou de bloco

int open(char *, int [, mode_t]);

– Retorno - índice na tabela de descritores de ficheiros.

– 1º parâmetro - localização do ficheiro

– 2º parâmetro - modos de acesso.
Bits juntos por disjunção 'I' (ex: o_RDWR I O_CREAT)

Nota: OR é bit a bit, não confundir com OR lógico 'II'

Modo	Acesso
O_RDONLY	Leitura apenas
O_WRONLY	Escrita apenas
O_RDWR	Leitura e escrita
O_APPEND	Acrescenta
O_CREAT	Cria ficheiro, se não existir

Nota: *creat(char* [, int] [, mode_t])* equivale a *open(, O_CREAT\O_WRONLY\O_TRUNC,)*

POSIX (5)

– 3º parâmetro (opcional) – permissões de acesso.

Elemento	Bandeira	Valor	Acesso
Dono ("owner")	S_IRWXU	0700	Leitura,escrita e execução
	S_IRUSR	0400	Leitura apenas
	S_IWUSR	0200	Escrita apenas
	S_IXUSR	0100	Execução apenas

- Nota1:
  Para Grupo ("group"), substituir U por G e USR por GRP.
  Para Outros ("other"), substituir U por O e USR por OTH.
- Nota2: processos com terminal associado abrem automaticamente 3 descritores de ficheiros:
  STDIN_FILENO (0), associado ao teclado
  STDOUT_FILENO (1), associado ao monitor
  STDERR_FILENO (2), associado ao monitor

POSIX (6)

- Abertura de um socket

int socket(int, int, int);

1º parâmetro – domínio de comunicação

2º parâmetro – tipo

3º parâmetro – protocolo de comunicação

- Abertura de um tubo

int pipe(int [2]);

1º parâmetro – localização onde são armazenados 2 descritores de ficheiro (leitura-índice 0, e escrita-índice 1 no tubo)

- Cópia de descritor de ficheiro

int dup(int);

Parâmetro – índice de descritor a copiar

Retorna – índice do novo descritor (o primeiro vazio)

POSIX (7)

B. Fecho de descritor de ficheiro

int close(int);

C. Funções de leitura/escrita

ssize_t read(int, void *, size_t);

Retorno – número de Bytes efectivamente lidos.

2º parâmetro – localização do local de instalação dos dados

3º parâmetro – dimensão do local de instalação dos dados

ssize_t write(int, const void *, size_t);

Retorno – número de Bytes efectivamente escritos.

POSIX (7)

D. Funções de reposicionamento

off_t lseek(int, off_t, int);

2º parâmetro – local de partida (valores iguais ao 3º parâmetro de fseek)

3º parâmetro – deslocamento

E. Manipulação do descritor de ficheiro

int fcntl(int, int, struck flock*);

2º parâmetro – comando de manipulação: POSIX apenas define

`F_DUPFD` : duplica descritor de ficheiro

`F_GETFD`/ `F_SETFD` : recolhe/altera bandeiras ("flags") do descritor

`F_GETFL`/ `F_SETFL` : recolhe/altera bandeiras ("flags") do ficheiro

`F_GETLK`/ `F_SETLK`/ `F_SETLKW` : determina dono/testa/tranca fechadura em zona de ficheiro

Atributos de ficheiros (1)

- Os atributos de ficheiros podem ser recolhidos num programa pela função `stat`, disponilibizada na biblioteca C e no Unix I/O.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int stat(char *, struct stat *);
int fstat(int, struct stat *);
```

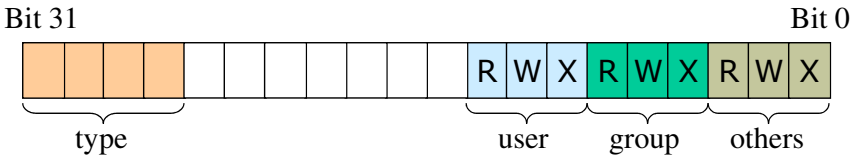
- Os atributos são amazenados na estrutura `stat`

Atributos de ficheiros (2)

```
struct stat {
  mode_t st_mode;      /* file type & mode & permissions */
  ino_t st_ino;        /* file inode number */
  dev_t st_dev,        /* device number (file system) */
        st_rdev;       /* device number for special files */
  nlink_t st_nlinks;   /* number of links */
  uid_t st_uid;        /* owner user ID */
  gid_t st_gid;        /* owner group ID */
  off_t st_size;       /* size in bytes, for regular files */
  time_t st_atime,     /* time of the last access */
         st_mtime,     /* time of the last modification */
         st_ctime;     /* time of the last status change */
  long st_blksize,     /* best I/O block size */
       st_blocks;      /* number of 512 byte blocks */
};
```


Atributos de ficheiros (3)

- Campo `st_mode` é um *bitset* do tipo de ficheiro e de permissões de acesso.



- Máscaras de permissões de acesso listadas no `open()`
- Máscaras de tipo de ficheiro (S_IFMT= 0170000)

Bandeira	Valor	Tipo ficheiro
S_IFREG	0100000	Regular
S_IFSOCK	0140000	Socket
S_IFLNK	0120000	Ligação simbólica
S_IFDIR	0040000	Directório
S_IFIFO	0010000	Tubo ou FIFO

Atributos de ficheiros (4)

- Exemplo: programa que testa se um ficheiro é um directório

```
struct stat sbuf;
int file = open(...);

if( stat( file, &sbuf ) == 0 )
  if( (sbuf.st_mode & S_IFMT) == S_IFDIR )
     printf("Directorio!\n");
```

- Macros são disponibilizadas em `<linux/stat.h>`

```
if( stat( file, &sbuf ) == 0 )
  if( S_ISDIR(sbuf.st_mode) )
     printf("Directorio!\n");
```